Spring 2021 ME/CS/ECE759 Final Project Report
University of Wisconsin-Madison

CPU Based Bitcoin Mining

Trevor Zachman-Brockmeyer

May 7, 2021

Abstract

Crypto currencies such as Bitcoin have recently begun to explode in both popularity and price [1]. In order for new Bitcoins to be introduced into the market, software must expend considerable resources in order to solve computationally expensive mathematical problems. These mathematical problems, which at their core are simple tasks that must be repeated many times, can be solved extremely efficiently using consumer level graphics processing units (GPUs). Due to Bitcoin's large spike in price, it has become increasingly profitable to utilize these consumer level GPUs in order to mine for Bitcoin. The high demand for GPUs has put significant pressure on the GPU market, resulting in their prices to skyrocket, and their stocks to diminish [2]. This has impacted both the profit margins of Bitcoin miners as well as everyday consumers who require GPU hardware for hobbies and work. With popular GPUs becoming scarce, is there another technology miners can turn to that can quickly solve the computational problems required to mine for Bitcoin? In this research, I utilize central processing unit (CPU) based parallelism techniques in order to see how comparable CPU Bitcoin mining is to the more popular GPU alternative.

** Note: no actual Bitcoin mining was done during this research. Data that was used for "mining" was either randomized or taken from Bitcoin blocks that were successfully mined years ago. **

Link to Final Project `git` repo:
https://euler.wacc.wisc.edu/trevorzachman-brockmeyer/me759-trevorzachman-brockmeyer/-/tree/master/FinalProject759

Contents

1. General information

- CS
- MS
- Trevor Zachman-Brockmeyer
- I release the ME759 Final Project code as open source and under a BSD3 license for unfettered use of it by any interested party.

2. Problem statement

In my original Final Project Proposal, I intended to create primarily GPU based Bitcoin miners. However, early on in my research, I had difficulties finding and successfully using a GPU C++ SHA-256 hashing implementation that I could port to Euler. Due to this complication and the time constraints of the project, I instead changed the focus of my research to purely CPU based Bitcoin mining. This change of focus to CPU powered mining allowed me to easily find and implement CPU C++ SHA-256 hashing implementations that were functional on Euler nodes.

I believe that exploring parallelized CPU Bitcoin mining is an even more interesting research topic as compared to GPU mining due to the current day controversies surrounding cryptocurrency mining straining the GPU hardware market. The mining of cryptocurrencies has become so profitable and the demand for GPUs has spiked so drastically that many average consumers are unable to buy GPUs for a reasonable price [2]. If we could design efficient mining algorithms on hardware other than consumer level GPUs, perhaps the GPU market could stabilize due to the decrease in demand.

This research seeks to answer the question "Can supercomputer clusters, using purely CPU hardware spread across multiple nodes, compete with common GPU bitcoin mining speeds?". In order to accomplish this, we design multiple iterations of CPU based bitcoin miners. These iterations span from naive serial algorithms, to OpenMP threaded work share, to OpenMPI processes communicating across multiple connected nodes. For each iteration of the miner, we gather throughput data in order to determine which implementation is fastest, and which parameters can be tuned in order to create a mining system that can hopefully compete against some modern day GPUs.

3. Solution description

Building of the mining framework: Bitcoin uses an 80 byte block header as the base data structure to feed into the SHA-256 hashing algorithm. This structure can be seen in the "block_header.h" file. Essentially, the structure consists of a version number, the hash of the previous chain block, the hash of the merkle root, a timestamp, a difficulty setting, and a modifiable nonce value. Along with this structure, it was also necessary to create a utility framework that handled creation and modification of the header struct, reformatting of hex strings, timestamps, and difficulty settings, and other general functionality that is needed throughout the miner. This utility framework can be seen in the "block_header.cpp" and "helpers.cpp" files.

SHA-256 hashing: The next step was to find a fast, easily implementable, and lightweight SHA-256 hashing implementation built in C/C++. The first algorithm found and implemented was the picosha2 hasher [3]. This was used during early stages of the research because of its easily accessible API built on functions that only required a header file to use. Later in development, we looked for a more efficient SHA-256 implementation. We found a popular open source Bitcoin miner whose SHA-256 hashing algorithm was built for speed [4]. By switching out the picosha2 for the sha2 of the cgminer as well as making some code verifications more efficient, we were able to increase the speed of our miner by 4 times. The picosha2 hasher can be found in the "picosha2.h" file and the cg hasher can be found in the "sha2.h" and "sha2.c" files.

Mining: Using the mining framework and the SHA-256 hashing algorithm, the actual mining can be performed. All mining implementations mentioned in the next paragraphs use the base mining process defined as the double SHA-256 hashing of the block header structure. You can find this mining algorithm in the "mine.h" and "mine.cpp" files. The "proof_of_concept_main.cpp" file uses the mining process to mine the first block ever mined for Bitcoin. This block was mined back in 2009 and since we know what the hash is supposed to be, we can use it as a proof of concept to check the integrity of our research's Bitcoin mining implementation.

Serial Miner: With the mining framework and hashing algorithm in place, we were able to create the first naive Bitcoin miner which was completely serial. No multithreading or multiprocessing was used in this miner. We were able to use the serial miner as a baseline to compare our parallelized implementations against. The serial miner uses a simple for loop that increments the nonce of the header, hashes the header, checks if a solution has been found, then repeats. The serial miner can be found in the "serial_miner.h", "serial_miner.cpp", and "serial_miner_main.cpp" files.

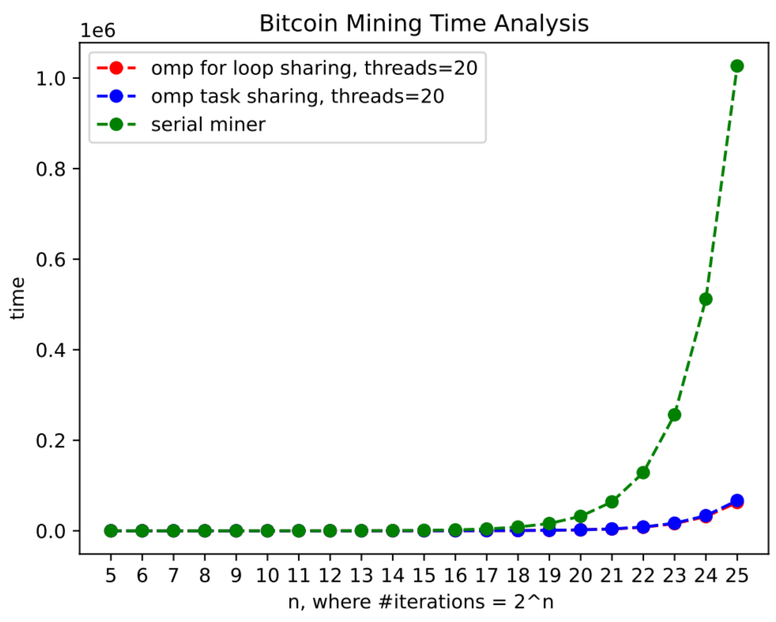
OpenMP Miner: With the serial miner baseline complete, the next step was to create a threaded miner using OpenMP. Due to the fine grained parallelism needed by mining, two versions of the OpenMP based miner were created. The first version utilized For Loop work sharing. This implementation is very similar to the serial miner, the difference being the for loop's work was shared between threads with a static schedule. A static schedule is best in this circumstance due to the workload being balanced between individual iterations of the loop. The second version used OpenMP's Task based work sharing. In this schema, one thread handles the creation of the block headers and the assignment of the nonces, while the rest of the worker threads use the finalized block headers in hashing calculations. The OpenMP For Loop Miner can be found in the "omp_for_miner.h", "omp_for_miner.cpp", and "omp_for_miner_main.cpp" files. The OpenMP Task Miner can be found in the "omp_task_miner.h", "omp_task_miner.cpp", and "omp_task_miner_main.cpp" files.

OpenMPI + OpenMP Miner: The final iteration of the CPU Bitcoin miner utilized both OpenMPI and OpenMP in order to maximize the speed of block header hashing. OpenMPI allowed the miner to issue work to multiple processes spread across multiple nodes in the Euler cluster. Each of these processes then parallelized further by spawning multiple OpenMP threads to share the work of individual mining attempts. OpenMPI facilitates communication between the processes using the master process of rank=0. Once a process finishes its share of the work, it reports its findings to the master process, who collects the results from all processes in order to analyze them to determine if a successful mining was completed or not. While waiting for worker processes to finish computations, the master process is able to complete its fair share of mining/hashing as well. The OpenMPI + OpenMP Bitcoin miner can be found in the "mpi_miner_main.cpp" file.

Data Analysis: With all iterations of the CPU Bitcoin miner fully functional, data could be collected and analyzed. First, the serial, OpenMP For, and OpenMP Task miners were analyzed using a varied number of mining attempts. Once a clear fastest algorithm was established between the three analyzed miners, it was used in conjunction with OpenMPI to create the final version of the Bitcoin miner. This final version was analyzed using varying numbers of Euler nodes, MPI processes, and OpenMP threads. The most efficient version of the miner with optimized parameters then had its SHA256 hashes per second compared to recorded GPU values.

4. Overview of results. Demonstration of your project

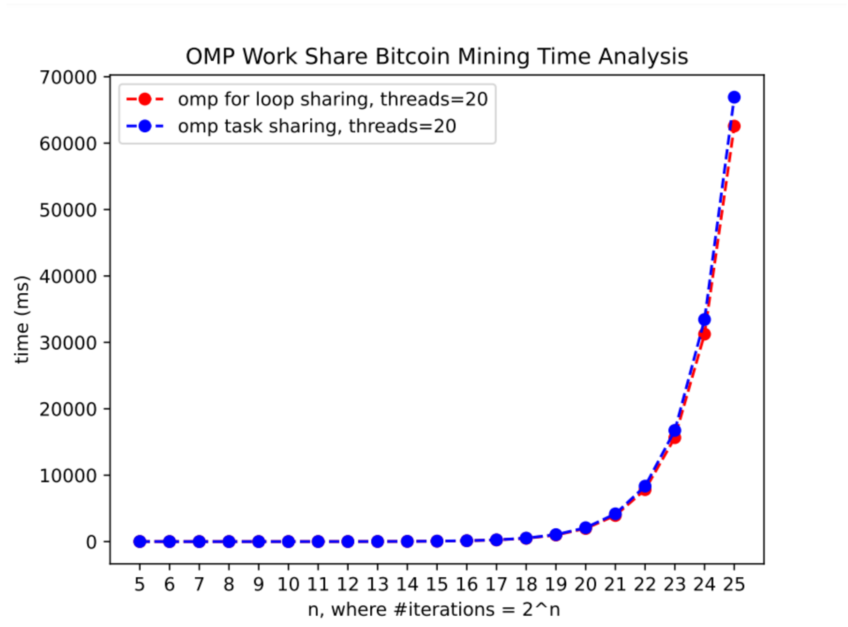
Serial Miner vs OpenMP Miners: The first data collected compared the naive serial version of the miner to the miners that were parallelized by OpenMP threads.



Serial miner (green) vs OpenMP For Loop Miner (red) vs OpenMP Task Miner (blue). X-axis is the number of iterations (mining attempts) in log base 2 form.

As clearly demonstrated in this graph, the naive serial implementation cannot handle anywhere close to the number of iterations that would be needed for efficient bitcoin mining. However, OpenMP parallelized mining shows promising results compared to the serial miner.

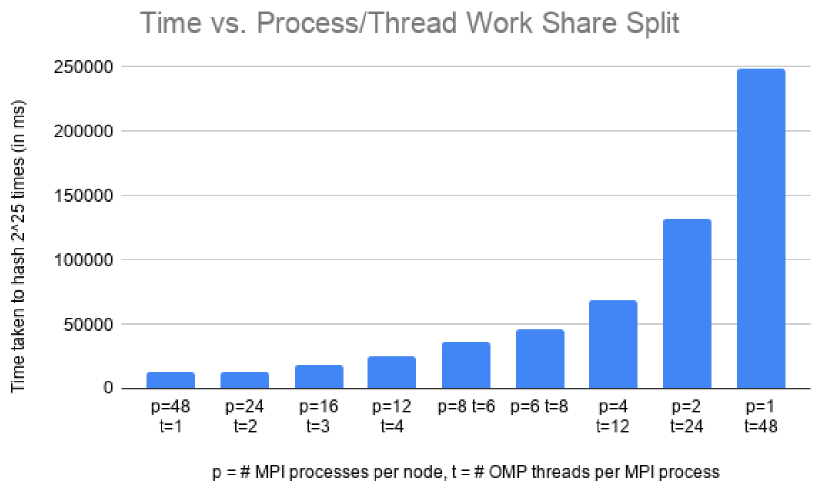
OpenMP For Loop Work Share vs OpenMP Task Work Share: The next data analyzed was the difference in speed between the two OpenMP work share options used: For Loop Work Share and Task Work Share.



OpenMP For Loop Miner (red) vs OpenMP Task Miner (blue). X-axis is the number of iterations (mining attempts) in log base 2 form.

Although the two variants are similar for low iteration counts, as the iterations increase, the For Loop variant begins to outperform the Task variant. This difference in speed only gets larger as the iterations count increases. The difference between the variants is likely due to worker threads waiting for the master thread to create tasks, or the master thread running out of work to do after finishing the creation of the tasks.

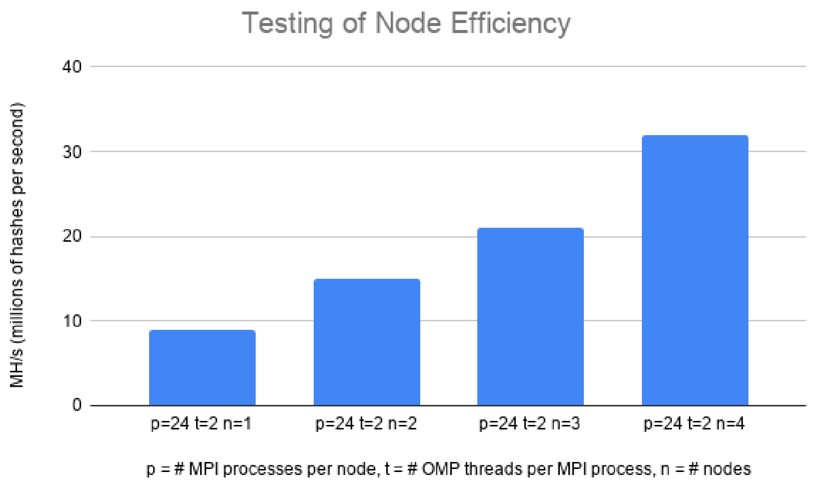
OpenMPI + OpenMP Parameter Optimization: With the knowledge of the OpenMP For Loop miner being faster than the Task based variant, we could then move on to the optimization of the OpenMPI + OpenMP miner. Euler allows 48 virtual CPUs to be used per node. These CPUs can be split between OpenMPI processes and OpenMP threads. Our next data analysis step sought to find the optimal balance between these two parallelizing methods.



X-axis: Different combinations of number of OpenMPI processes and OpenMP threads per process. Y-axis: time taken for 2^25 iterations.

These results show that using most of the 48 CPUs for OpenMPI processes yielded the fastest hashing times. The optimal combination found was the use of 24 OpenMPI processes each running 2 OpenMP threads.

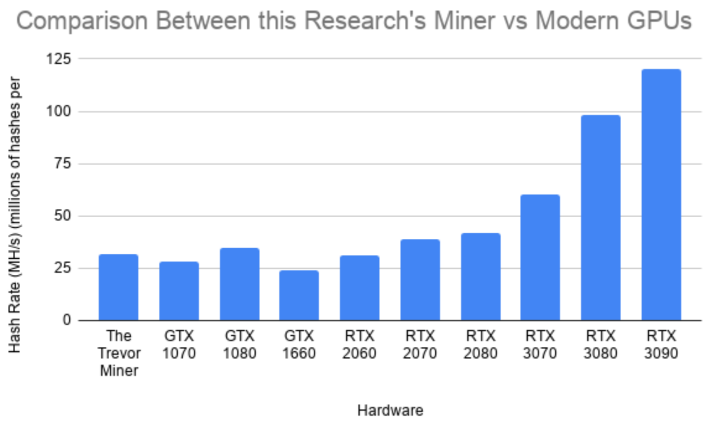
Optimized Miner with Varying Nodes: Euler allows 4 nodes to be used for a batch job. In order to make sure the implemented MPI message system is working as intended and not impacting performance as nodes are added, we collected time analysis data on constant workloads spread across varying numbers of nodes.



X-axis: OpenMPI + OpenMP miner using different numbers of nodes. Y-axis: hashes completed per second (in millions).

This demonstrates a clear linear increase in hash speed as node size increases. The doubling of hash speed from n=2 to n=4 is a good indicator that our mining implementation scales linearly with total processing power available.

Final Optimized Miner vs GPUs: With our final optimized CPU miner established (4 nodes, 24 OpenMPI processes per node, and 2 OpenMP threads per process), we could test its hashrate against GPUs of varying processing powers.



Graph: Comparing the fastest iteration of our CPU miner (4 nodes, 24 processes per node, 2 threads per process) against modern GPU hashing rates. Y-axis scale is in millions of hashes per second.

We can see from this graph that our mining implementation can beat or at least compete with many older model GPUs. However, once we start making comparisons to modern, high end GPUs, our miner's hashrate cannot keep up. Along with this, our miner uses 4 nodes to reach the 32 MH/s compared to these single hardware GPUs that can run on a single node.

Results Discussion Serial vs Threaded: Our results show that a naive, serial Bitcoin miner cannot achieve hashrates anywhere close to a value that is useful for real life situations. Our results also demonstrate that the serial miner is able to keep pace with the threaded OpenMP miners until around

2^15 iterations. After this point (2^16 iterations and onwards), the cost of creating and maintaining OpenMP threads is well worth the hashing speed the parallel methods provide.

Results Discussion Task vs For Loop OpenMP Work Share: As the number of iterations increases, the OpenMP For Loop based miner begins to outperform the OpenMP Task based miner. At around 2^20 iterations, the For Loop miner is noticeably more efficient than its task based counterpart. This difference in performance could be due to the overhead cost of the creation and upkeep of OpenMP tasks, or due to the master thread generating tasks too quickly, and running out of things to do. Because OpenMP task generation is generally extremely quick, the slowdown is likely not due to the worker threads idling while waiting for tasks to be made available.

Results Discussion Process vs Thread Work Split: Our results showed that greater hashing speed was achieved when virtual CPUs were being used by primarily OpenMPI processes as opposed to OpenMP threads. The best results came from 24 OpenMPI processes and 2 OpenMP threads per process, however, using just 48 OpenMPI processes and no OpenMP threads achieved similar speeds. This could be due to the fact that the processes do not share virtual memory spaces but the threads do. That shared memory could lead to some sort of memory conflict slowing down the threads during the hashing process. Further research would have to dive deeper into the mining algorithm and see if there are any glaring memory issues.

Results Discussion CPU Miner vs GPU Miners: Our final results showcased our fastest CPU miner vs GPU competitors. Our miner, using 4 Euler nodes, 24 processes per node, and 2 threads per process, achieved a hashrate of 32 MH/s. This hashrate was calculated by taking the number of hash-iterations divided by the time taken in seconds. This rate is faster than or extremely similar to the hashrates found for NVIDIA GPUs: GTX 1070, GTX 1080, GTX 1660, and the RTX 2060. However, once you start comparing the 32 MH/s hashrate to more modern/powerful NVIDIA GPUs such as the RTX 2080, RTX 3070, RTX 3080, and RTX 3090, the GPU miners begin to pull much higher hashrates [5]. This is primarily due to GPU architecture being built specifically for high numbers of mathematical computations, which are required for the SHA-256 hashing algorithm. In the end, the computational power of modern GPUs outclasses modern CPUs when it comes to cryptocurrency mining.

5. Deliverables:

Main Deliverables: The main deliverables of this research are multiple, fully functioning, CPU based Bitcoin miners. Versions of the miner include (in order of increasing mining speed) the serial miner, the OpenMP Task miner, the OpenMP For Loop miner, and the OpenMPI + OpenMP For Loop miner. These miners can be tested using the demo script present within the codebase. More info on the demo script can be found in the "Run Demo" paragraph further down in this section.

Code Layout: The main directory of the git repo contains all the code used for all versions of the CPU Bitcoin miner. Each version of the miner usually consists of files following the pattern of "xxx_miner.cpp", "xxx_miner.h", "xxx_miner_main.cpp", and a corresponding "xxx_miner.sh" Slurm script file. Each miner also utilizes functions and structs in the "mine.h", "mine.cpp", "helpers.h", "helpers.cpp", "block_header.h", "block_header.cpp", "sha2.h", and "sha2.c" files. There is also a subdirectory titled “outputs” that houses many of the “.out” and “.err” files created by Euler Slurm scripts during the collecting of data for this paper.

Run Demo: I created a demo script that runs the proof of concept as well as all versions of the miner on a small number of hash iterations. With the git repository on Euler, simply issue the command “sbatch demo.sh”. The script will handle all module installs, compiles, cleans, and logging. No prior work is necessary for the script to run. The script usually takes around 30-40 seconds to completely run and finish cleanup. Once the script finishes, there should now be a “demo.out” file in the same directory as the script (where all the code files are). This log file details the proof of concept as well as the running of the miner including hashrate, number of iterations ran, time taken, numbers of threads/processes/nodes used if applicable, the target hash we are aiming for, examples of the calculated hashes, MPI processes communicating with each other, and a message indicating if a theoretical Bitcoin was successfully “mined” or not. Note: although the final miner utilizes 4 Euler nodes to reach the 32 MH/s hashrate, the demo script only uses 1 to ensure the script runs without Euler restricting run time due to the large resource demand.

(If for some reason the “demo.sh” script fails to run, there is a “demo_example.out” log file in the main code directory that shows what the demo script would have produced if it ran correctly.)

6. Conclusions and Future Work

This research has showcased that it is indeed possible to create purely CPU based crypto mining software that can hold its own against its more popular GPU counterparts. Having said this, there are some limitations to these hashing speed results that must be addressed in future work.

Multiple Nodes vs One GPU: The final version of the miner created by this research uses 4 Euler nodes running at maximum occupancy (all 48 virtual CPUs working on each separate node). This multi-node miner was compared to singleton GPUs which could run on a single Euler node. In order for the comparisons to be conducted on a more level playing field, we could instead use the maximum hashrate found for the CPU miner using only 1 Euler node. Doing this, our miner’s hashrate drops to around 9 MH/s, which is well below the GPU based hashrates compared to in our data section.

Power Usage Comparisons: Although hashrate is an extremely important statistic when designing and comparing Bitcoin miners, another equally important parameter is the power consumption of the system. While high hashrates positively affect the income of the mining process, the power used to achieve these hashrates adds expense that will ultimately have to be subtracted from the income made. This research did not investigate the power consumption of our finalized Bitcoin miner. Further research into determining this would be incredibly useful in determining the overall efficiency of a CPU miner vs a GPU based one.

Dedicated Hardware Miners: Another comparison not made in this research is the comparison between our CPU miner and hardware that is designed specifically for bitcoin mining. These comparisons were left out of this research for two reasons. First, we wanted to compare mining methods that are available to anyone who has access to a computer/supercomputing cluster and the hardware within them. While GPU vs CPU comparisons fit this criteria, dedicated hardware miners do not. Secondly, the hashrates achieved by these dedicated hardware systems are astronomically larger than the rates achieved by CPUs or even GPUs. For example, the “Antminer S19 Pro 110 TH” - with a price tag of $15,000 - 110 TH/s (trillions of hashes per second) [6].

References

[1] https://www.coindesk.com/price/bitcoin
[2] https://www.nytimes.com/2018/05/08/technology/gpu-chip-shortage.html
[3] https://github.com/okdshin/PicoSHA2
[4] https://github.com/ckolivas/cgminer
[5] https://www.nicehash.com/blog/post/nvidia-and-amd-graphics-card-oc-settings-for-mining
[6] https://compassmining.io/hardware/bitmain/antminer-s19-pro-110-th